Lab 3 SQL Injection and JDBC

Table of Contents

Goal:

After this lab, student should know how to

1. How to access database through JDBC
2. How to prevent SQL injection.

Students should master the following specifics after the lab:

1. Use JDBC API to talk to database
2. Prepared statement
3. Result set
4. SQLSTATE
5. Transactions

References

1. MySQL reference

https://dev.mysql.com/doc/refman/8.0/en/sql-statements.html

2. JDBC API Documentation

http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/

Summary of the lab tasks

We will develop some bank functions using Java and database. We need at least 3 software

A. Java Development Kit (JDK)
B. Java Develop Environment (Eclipse, VSCode, vim, emacs etc)
C. Java Database Connector (JDBC driver)

We will be implementing 6 functions in the file JDBCLab.java

A. connectDB()
B. disconnect()
C. displayAccount()
D. transfer()
E. transfer_NoSQLInjection()
F. newConnectDB()

Lab Report:

- You need to include the screen output for running each function as you work on them one by one.
- You also need to submit the final version of JDBCLab.java after you complete all 6 functions.

Lab Report template:

- For your convenience, a report template (Lab3_JDBC_Report_template.doc) has been created for you. Please use it.

1. Set up the database and the programming environment

1.1. Create the following table in your database
Lab3_account (account_number char(10) primary key, balance double not null)

1.2. Insert at least 3 records in the table above.
Insert into Lab3_account values('A001', 100);
Insert into Lab3_account values('B001', 999);
Insert into Lab3_account values('C001', 99.99);

1.3. There are 3 options of programming environment to choose from. **You need to decide which option is best to you based on your own experiences. Please discuss these options with other students in the class.**

Option1: Do everything on your local computer. Don't use wopr.csl.mtu.edu at all. This is the recommended option.

You need to have the following on your local computer:

- Code editor (notepad, Visual Studio Code, Eclipse etc)
  - If you like to use VScode, see https://code.visualstudio.com/docs/java/java-tutorial
- VPN (if you are not on campus network)
- Java SDK
- JDBC Driver
  - Download JDBC driver (download from canvas or Internet)
  - If you use command line, you will need to use command options when you execute your java code.
  - If you use IDE, you need to configure it in your IDE. Here is how to configure it in Eclipse
    1) Right click on your project.
    2) Select Build Path.
    3) Click on Configure Build Path.
    4) Click on Libraries and select Add External JARs.
    5) Select the jar file from the required folder.
    6) Click and Apply and Ok.

Option2: Do everything remotely on wopr.csl.mtu.edu. Wopr.csl.mtu.edu has all required software installed.

- Remote login (ssh, putty) to terminal to wopr.csl.mtu.edu
- Use text editor (vim, emacs) on the terminal to wopr.csl.mtu.edu .
- Compile (javac) and run (java) your java code on the terminal to wopr.csl.mtu.edu
  - The JDBC driver has been installed on wopr.csl.mtu.edu. The driver is /usr/share/java/mysql-connector-java.jar
  - I recommend you to have two terminals open at the same time. One for text editing, one for compiling and running your code.
- You need to transfer the file (File Zilla) from wopr to your local computer, so it can be included to your report

Option3: Edit the file locally, Compile and run the code on wopr.csl.mtu.edu

- Edit the file locally with a local code editor (notepad, Visual Studio Code, Eclipse etc)
- Transfer the file to wopr.csl.mtu.edu everytime you made a change to the source code (File Zilla) before you compile and run the code
- The rest of the steps is same as option1

1.4 Get the programming environment ready.

- Install all software based on the option you have selected.
  - If you are going to compile and run your code on your local computer, **download the mysql JDBC driver mysql-connector-java.jar** from Canvas to the above folder Lab3
- Create a folder on your local computer (or wopr.csl.mtu.edu) called **Lab3.**
- Start VPN if you are not on campus and choose to do everything on your local computer.

2. Use JDBC functions to connect to database

Create a java source file JDBCLab.java under the folder Lab3. You may type or copy the following lines to JDBCLab.java. Replace the argument in DriverManager.getConnection() with your own information. You are going to hardcoded the password here just for this problem. Later in the lab, you will learn how to prompt user to enter the password. For safety, please consider change your password before and after this problem just in case someone sees your password. In the report, replace it with string of XXXX.

```
import java.sql.*;

public class JDBCLab {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        public int connectDB(){
                try {
                        conn = DriverManager.getConnection(
                           "jdbc:mysql://classdb.it.mtu.edu/YOURDBNAME",
                           "YOURACCOUNT",
                           "YOURPASSWORD");
                        System.out.println("Connected to the database!");
                } catch (SQLException e) {
                        System.out.println(e.getMessage());
                        e.printStackTrace();
                        return 1;
                }
                return 0;
        }

        public void disconnect(){
                try {
                        conn.close();
                         System.out.println("Disconnected from the database!");

                }
                catch (SQLException ex){
                        System.out.println("SQLException: " +
                                           ex.getMessage());
                        System.out.println("SQLState: " + ex.getSQLState());
                        System.out.println("VendorError: " +
                                           ex.getErrorCode());
                }
        }

        public static void main(String args[]){
                JDBCLab dblab = new JDBCLab();

                dblab.connectDB();
                dblab.disconnect();

        }
}
```

If you have configured the JDBC driver in your IDE, simply click the run button to run your application.

If you want to run the code on the terminal, please following this steps.

A. In the termial, compile JDBCLab.java.

```
javac JDBCLab.java
```

B. In the terminal, run JDBCLab.class

After the code has been compiled successfully, to run your java code, you need to use option "-cp" to specify where is the class file and where is the JDBC driver. Optiopn "-cp" stands for CLASS PATH). Dot (.) means the current directory. The command is slightly different based on where you run the code.

- On wopr.csl.mtu.edu

```
java -cp .:/usr/share/java/mysql-connector-java.jar JDBCLab
```

- On your own Linux/Mac. This command assumed that you had put the driver under Lab3, the same directory as JDBCLab.java

```
java -cp .:mysql-connector-java.jar JDBCLab
```

- On You your own Windows terminal. Notice the quotation marks and the semicolor.

```
java -cp ".;mysql-connector-java.jar" JDBCLab
```

You should see a line

```
Connected to the database!
Disconnected from database!
```

3. Retrieve data from database

Let's add more functions to JDBCLab.java. First, implement function displayAccount() and call the function in the main method.

```
    public void displayAccount(){

            try {
                    stmt = conn.createStatement();
                    rs = stmt.executeQuery("SELECT account_number,balance
                                            FROM Lab3_account");
                    while (rs.next() ) {
                            System.out.println(rs.getString(1)+ ","+
                                               rs.getString(2));
                   }
            }
            catch (SQLException ex){
                    System.out.println("SQLException: " +
                                       ex.getMessage());
                    System.out.println("SQLState: " + ex.getSQLState());
                    System.out.println("VendorError: " +
                                       ex.getErrorCode());
            }
    }
```

Compile and run your code. You shall see the output:

```
Connected to the database!
A001,70
B001,999
C001,99.99
Disconnected from database!
```

4. Update the data in the database using transaction

Next, let's add function transfer() method. Type the code as below and Run the program. Please pay attention how transaction is used.

```
import java.util.Scanner;

public int transfer(String from_account_number, String to_account_number, double amount)
throws SQLException{
        Statement stmt = null;
        ResultSet rs = null;
        int rowcount;

        try {
            // start transaction
            conn.setAutoCommit(false);
            conn.setTransactionIsolation(
                Connection.TRANSACTION_SERIALIZABLE);
        } catch (SQLException e) {
            e.printStackTrace();
            return 0;
        }

        try {

            //add code here later to check the balance of from_account_number
            //if it is smaller than amount, rollback the transaction.


            String sqlstr;
            stmt = conn.createStatement();


            sqlstr = "update Lab3_account set balance = balance - " + amount
                    + " where account_number = '" + from_account_number + "'";


            rowcount = stmt.executeUpdate(sqlstr);
            System.out.println("deduct money from account "+ from_account_number + ": " +
rowcount + " rows has been updated");

            sqlstr = "update Lab3_account set balance = balance + " + amount
                    + " where account_number = '" + to_account_number + "'";

            rowcount = stmt.executeUpdate(sqlstr);
            System.out.println("save money to account "+ to_account_number + ": " +
rowcount + " rows has been updated");


            conn.commit();
        }
        catch (SQLException ex){
            // handle any errors
            System.out.println("SQLException: " + ex.getMessage());
            System.out.println("SQLState: " + ex.getSQLState());
            System.out.println("VendorError: " + ex.getErrorCode());
             conn.rollback();
        }

        return 1;
    }
```

```
    public static void main(String args[]){

        JDBCLab dblab = new JDBCLab();
        dblab.connectDB();
        dblab.displayAccount();
```

```
        String from_account, to_account;
        Double balance;
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the account to tranfer the money from:");
        from_account = input.nextLine();
        System.out.println("Enter the account to tranfer the money to:");
        to_account = input.nextLine();
        System.out.println("Enter the amount to withdraw:");
        balance = input.nextDouble();
        System.out.println("tranfering $" + balance + " from " + from_account + " to " +
to_account);
        dblab.transfer(from_account, to_account, balance);

        dblab.displayAccount();
        dblab.disconnect();
    }
```

Compile and run you code and make sure it works as you expected.

5. Have a taste of SQL Injection

Let's use the transfer function to give everyone extra $100.

When the prog asks for from account, type who cares?
When the prog asks for to account, type `X' or 1=1 #toeveryone`
When the prog asks for to amount, type 1000000

Run your program and see what happens to the balance of all accounts.

6. Use prepared statement to prevent SQL injection.

Create a new method transfer_NoSQLInjection() as below. Pay attention to how question mark (?) is used in the place of user input data. In your main function, call this new transfer method.

```
public int transfer_NoSQLInjection(String from_account_number, String to_account_number,
double amount){
    PreparedStatement stmt = null;
    ResultSet rs = null;
    int rowcount;

    try {
    // start transaction
    conn.setAutoCommit(false);
    conn.setTransactionIsolation(
        conn.TRANSACTION_SERIALIZABLE);
    } catch (SQLException e) {
        e.printStackTrace();
        return 0;
    }

    try {
      String sqlstr;

      sqlstr = "update Lab3_account set balance = balance - ? where account_number = ? ";
      stmt = conn.prepareStatement(sqlstr);
      stmt.setDouble(1, amount);
      stmt.setString(2, from_account_number);

      rowcount = stmt.executeUpdate();
      System.out.println("deduct money from account "+ from_account_number + ": " +
rowcount + " rows has been updated");


      sqlstr = "update Lab3_account set balance = balance + ? where account_number = ? " ;
      stmt = conn.prepareStatement(sqlstr);
      stmt.setDouble(1, amount);
      stmt.setString(2, to_account_number);
      rowcount = stmt.executeUpdate();
      System.out.println("save money to account "+ to_account_number + ": " + rowcount +
" rows has been updated");


      conn.commit();
    }
    catch (SQLException ex){
      // handle any errors
      System.out.println("SQLException: " + ex.getMessage());
      System.out.println("SQLState: " + ex.getSQLState());
      System.out.println("VendorError: " + ex.getErrorCode());
    }

    return 1;
}
```

Now play the same trick as you did before and see what happens:

When the prog asks for from account, type who cares?

When the prog asks for to account, type `X' or 1=1 #toeveryone`

When the prog asks for to amount, type 100

7. Rollback the transaction when something is wrong

Modify function transfer_NoSQLInjection(). When there is no sufficient balance or the number of rows that has been updated is not 1, rollback the transaction.

```
String sqlstr;
sqlstr = "select balance  from Lab3_account where account_number = ? ";
stmt = conn.prepareStatement(sqlstr);
stmt.setString(1, from_account_number);
rs = stmt.executeQuery();
if (rs.next()) {
      double balance = rs.getDouble(1);
      if (balance < amount) {
            System.out.println("Insufficient balance");
            conn.rollback();
            return 0;
      }
} else {
      System.out.println("Account does not exist ");
      conn.rollback();
      return 0;
}

…

// TODO by students: check row count after each update,…
```

8. (Optional) Read password from the console

Last, let's create our newConnectDB() function to prompt user to enter username and password. Please notice Console. readPassword() function does not echo the input, so other people will not see what you type. Modify your main to call this newConnectDB(). Replace your password in connectDB() with XXX. Then Compile and run your code to transfer $100 from one account to another account.

```
import java.io.Console;

public int newConnectDB(){
     try {
         String username=null;
         char[] password=null;
         Console console = System.console();
         if (console == null) {
            System.out.println("console is null. Run the program in terminal");
            return 1;
         }
         username = console.readLine("Please enter your name:");
         password = console.readPassword("Please enter your password:");

         conn = DriverManager.getConnection( "jdbc:mysql://classdb.it.mtu.edu/"+
username, username, String.valueOf(password));
         System.out.println("Connected the the database!");
     } catch (SQLException e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
         return 1;
     }
     return 0;
}
```